CPE 344
Digital System Design

Dr. Muhammad Naeem Awais

Room # 8, B-Block

COMSATS University Islamabad, Lahore Campus

About the Course

1	Course Title	Digital System Design
2	Course Code	CPE344
3	Credit Hours	4 (3,1)
4	Prerequisites	Digital Logic Design
5	Semester	Fall 2025
6	Resource Person/Lab Engineer	Dr. Muhammad Naeem Awais/Engr. Mr. Moazzam Ali Sahi
7	Contact Hours (Theory)	3 hours per week
8	Contact Hours (Lab)	3 hours per week
9	Office Hours	08:30 - 16:30 (weekdays)

CPE-344 Digital System Design

Topics to Be Covered:

1. Introduction to Digital Systems
2. Combinational Logic Design
3. Sequential Logic Design – Controllers
4. Datapath Components
5. Register-Transfer Level (RTL) Design
6. Optimizations and Tradeoffs
7. Programmable Logic Arrays (PAL), Programmable Arrays Logic (PLA)
8. Physical Implementation

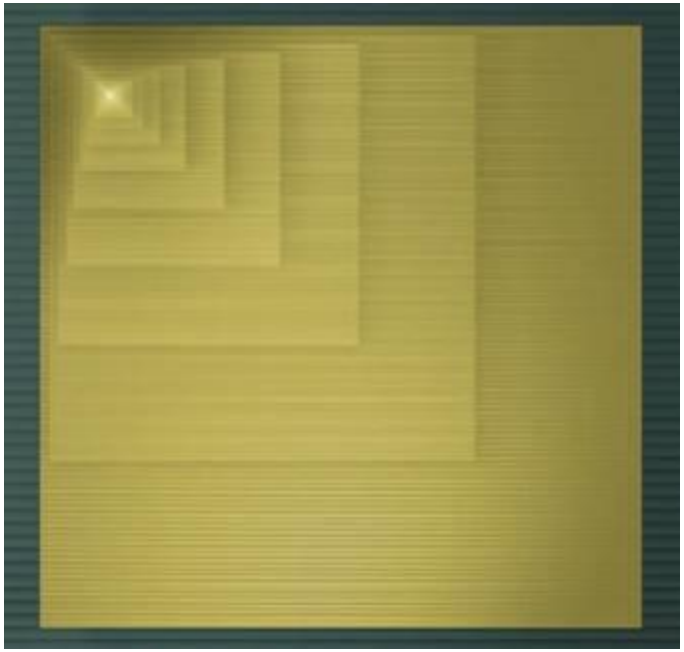
Course learning Outcomes

Theory CLOs:

- Design of advanced combinational and sequential logic-based systems using the classical principles of digital logic design. (C5-PLO3)
- Design of digital systems in a hierarchical and top-down manner using register-transfer level (RTL) approach and circuits to optimize their performance and ensure reliability, integrating techniques like concurrency and trade-offs in the physical implementation using RTL design. (C5-PLO3)

Lab CLOs:

- Design the digital systems based on HDL modeling techniques using VHDL. (C5-PLO3)
- Reproduce the response of the designed digital systems using the software tool and hardware platform. (P3-PLO5)
- Demonstrate proficiency in FPGA architecture and design flow for system synthesis and optimization while showing strong communication skills in preparing reports for complex engineering challenges. (A3-PLO10)



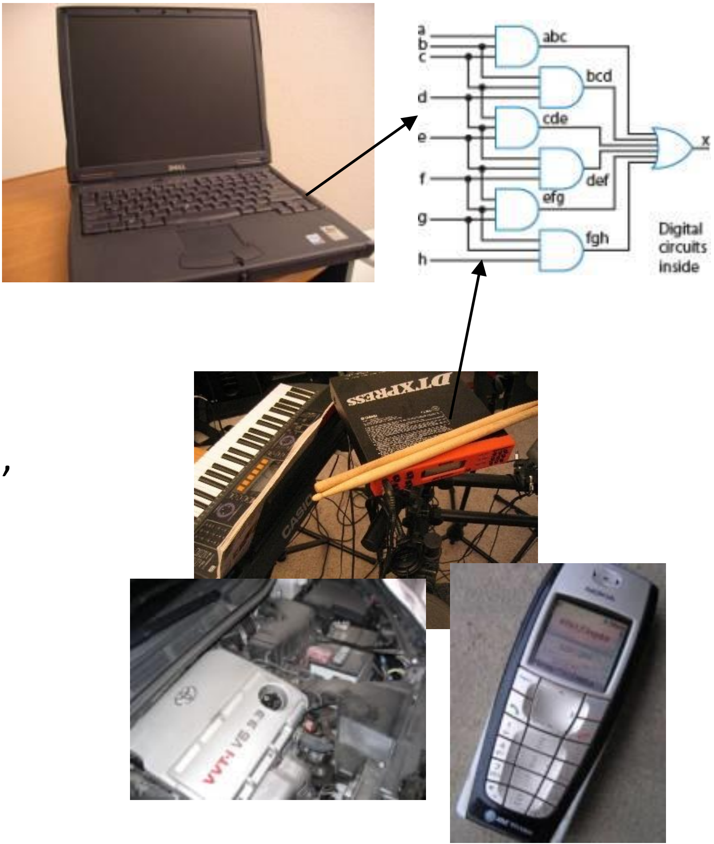
Digital System Design

Chapter 1: Introduction

Slides to accompany the textbook *Digital Design*, First Edition,
by Frank Vahid, John Wiley and Sons Publishers, 2007.
http://www.ddvahid.com

Why Study Digital Design?

- Look “under the hood” of computers
  - Solid understanding --> confidence, insight, even better programmer when aware of hardware resource issues
- Electronic devices becoming digital
  - Enabled by shrinking and more capable chips
  - Enables:
    - Better devices: Better sound recorders, cameras, cars, cell phones, medical devices,...
    - New devices: Video games, ...
  - Known as “embedded systems”
    - Thousands of new devices every year
    - Designers needed: Potential career direction



Satellites

Portable music players

Cell phones

DVD players

Cameras

Video recorders

TVs

Musical instruments

???

1995 1997 1999 2001 2003 2005 2007

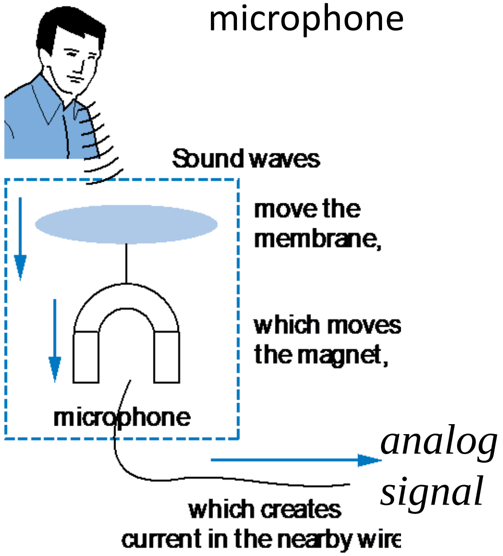
- Years shown above indicate when digital version began to *dominate*
  - (*Not* the first year that a digital version appeared)

Note: Slides with animation are denoted with a small red "a" near the animated items

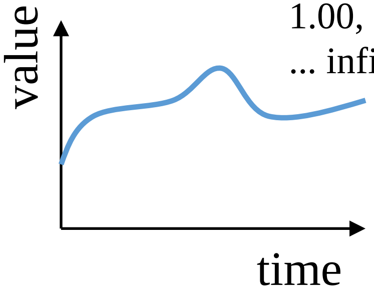
What Does “Digital” Mean?

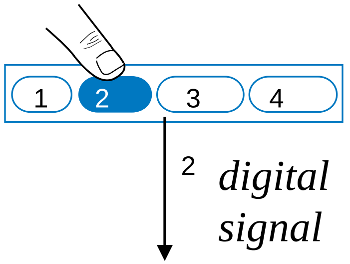
- Analog signal
  - Infinite possible values
    - Ex: voltage on a wire created by microphone

- Digital signal
  - Finite possible values
    - Ex: button pressed on a keypad

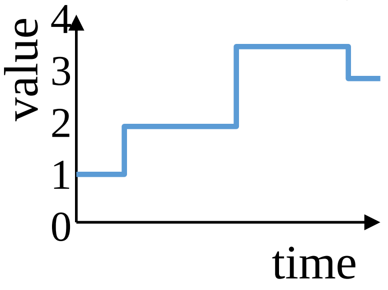


Possible values:
1.00, 1.01, 2.0000009,
... infinite possibilities



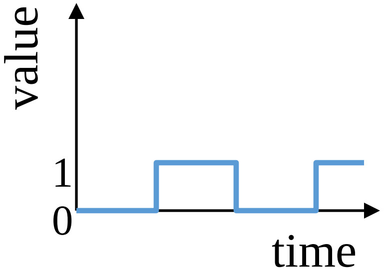


Possible values:
0, 1, 2, 3, or 4.
That’s it.



Digital Signals with Only Two Values: Binary

- ***Binary*** digital signal -- only *two* possible values
  - Typically represented as **0** and **1**
  - One *b*inary dig*it* is a ***bit***
  - We'll only consider *binary* digital signals
  - Binary is popular because
    - Transistors, the basic digital electric component, operate using *two* voltages (more in Ch. 2)
    - Storing/transmitting one of *two* values is easier than three or more (e.g., loud beep or quiet beep, reflection or no reflection)

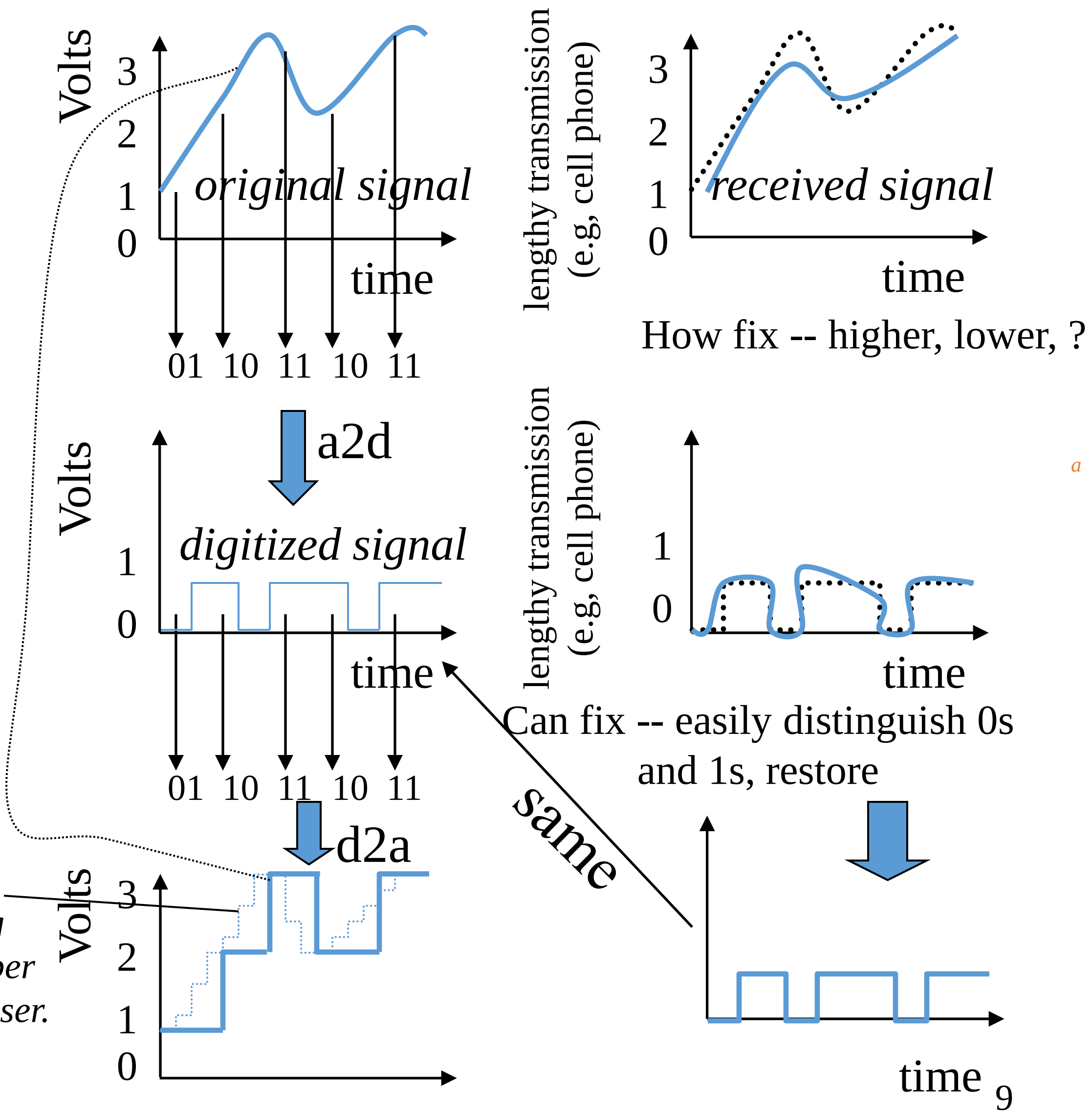


Example of Digitization Benefit

- Analog signal (e.g., audio) may lose quality
  - Voltage levels not saved/copied/transmitted perfectly
- Digitized version enables near-perfect save/cpy/trn.
  - “Sample” voltage at particular rate, save sample using bit encoding
  - Voltage levels still not kept perfectly
  - But we can distinguish 0s from 1s

Let bit encoding be:
1 V: “01”
2 V: “10”
3 V: “11”

Digitized signal not perfect re-creation, but higher sampling rate and more bits per encoding brings closer.



How fix -- higher, lower, ?

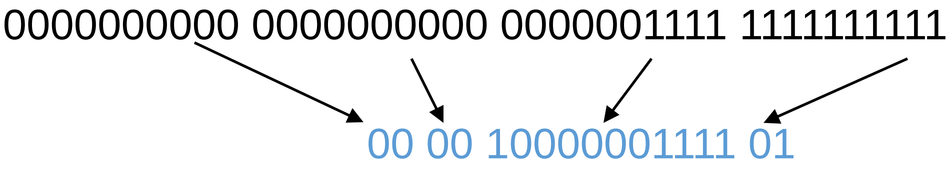
Can fix -- easily distinguish 0s and 1s, restore

Digitized Audio: Compression Benefit

- Digitized audio can be compressed
  - e.g., MP3s
  - A CD can hold about 20 songs uncompressed, but about 200 compressed
- Compression also done on digitized pictures (jpeg), movies (mpeg), and more
- Digitization has many other benefits too

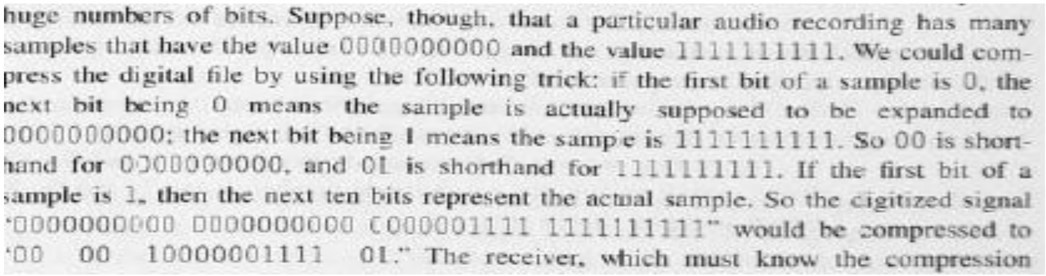
Example compression scheme:

00 --> 0000000000
01 --> 1111111111
1X --> X



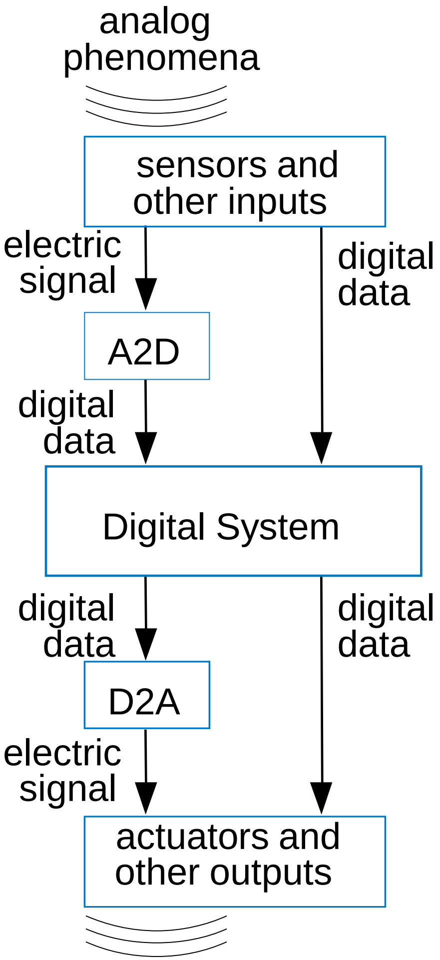
0000000000 0000000000 0000001111 1111111111

00 00 10000001111 01

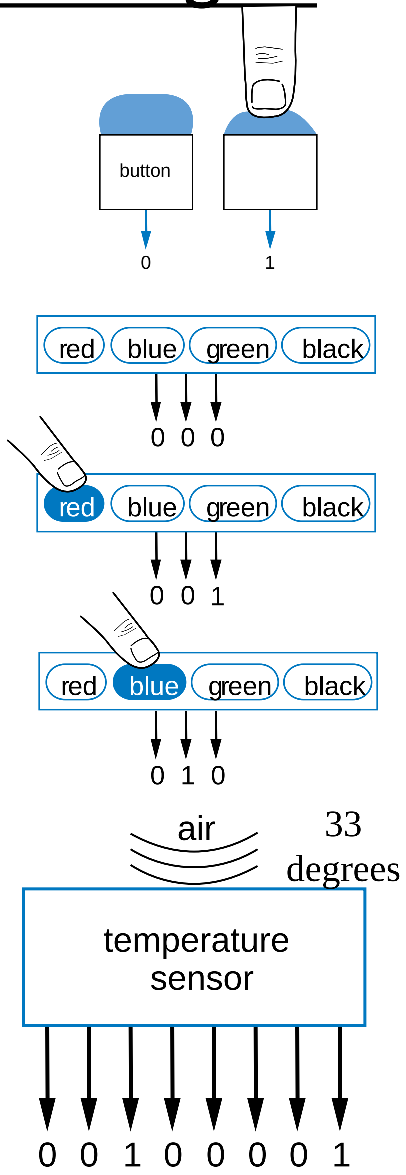


huge numbers of bits. Suppose, though, that a particular audio recording has many samples that have the value 0000000000 and the value 1111111111. We could compress the digital file by using the following trick: if the first bit of a sample is 0, the next bit being 0 means the sample is actually supposed to be expanded to 0000000000; the next bit being 1 means the sample is 1111111111. So 00 is shorthand for 0000000000, and 01 is shorthand for 1111111111. If the first bit of a sample is 1, then the next ten bits represent the actual sample. So the digitized signal "0000000000 0000000000 0000001111 1111111111" would be compressed to "00 00 10000001111 01." The receiver, which must know the compression

How Do We Encode Data as Binary for Our Digital System?



- Some inputs inherently binary
  - Button: not pressed (0), pressed (1)
- Some inputs inherently digital
  - Just need encoding in binary
  - e.g., multi-button input: encode red=001, blue=010, ...
- Some inputs analog
  - Need analog-to-digital conversion
  - As done in earlier slide -- sample and encode with bits



Number Systems

- Decimal
- Binary
- Octal
- Hexadecimal

Number Representation

- **Numbers**
    - ➢Integer (Fixed Point) Numbers
        - ❖Unsigned
        - ❖Signed
    - ➢Real (Floating Point) Numbers
        - ❖Unsigned
        - ❖Signed

Signed Number Representation

- Negative numbers are represented as
  - One's complement
  - Two's complement
  - Sign-Magnitude

How to Encode Text: ASCII, Unicode

- ASCII: 7- (or 8-) bit encoding of each letter, number, or symbol
- Unicode: Increasingly popular 16-bit bit encoding
  - Encodes characters from various world languages

Symbol	Encoding
R	1010010
S	1010011
T	1010100
L	1001100
N	1001110
E	1000101
0	0110000
.	0101110
<tab>	0001001

Symbol	Encoding
r	1110010
s	1110011
t	1110100
l	1101100
n	1101110
e	1100101
9	0111001
!	0100001
<space>	0100000

Question:

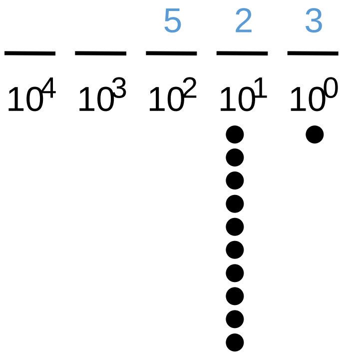
What does this ASCII bit sequence represent?

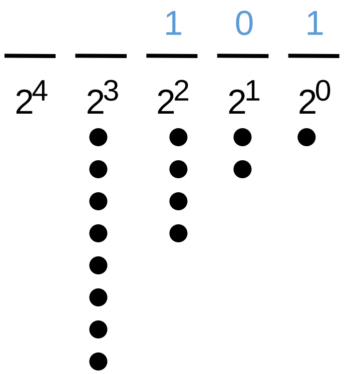
1010010 1000101 1010011 1010100

R E S T

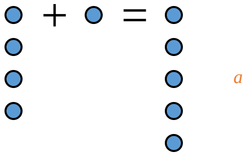
How to Encode Numbers: Binary Numbers

- Each position represents a quantity; symbol in position means how many of that quantity
  - Base ten (***decimal***)
    - Ten symbols: 0, 1, 2, ..., 8, and 9
    - More than 9 -- next position
      - So each position power of 10
    - Nothing special about base 10 -- used because we have 10 fingers
  - Base two (***binary***)
    - Two symbols: 0 and 1
    - More than 1 -- next position
      - So each position power of 2





Q: How much?



4 + 1 = 5

How to Encode Numbers: Binary Numbers

- Working with binary numbers
  - In base ten, helps to know powers of 10
    - one, ten, hundred, thousand, ten thousand, ...
  - In base two, helps to know powers of 2
    - one, two, four, eight, sixteen, thirty two, sixty four, one hundred twenty eight
      - (Note: unlike base ten, we don't have common names, like "thousand," for each position in base ten -- so we use the base ten name)
    - Q: count up by powers of two

___	___	___	___	___	___	___	___	___	___
2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
___	___	___	___	___	___	___	___	___	___
512	256	128	64	32	16	8	4	2	1

512 256 128 64 32 16 8 4 2 1 a

Converting from Decimal to Binary Numbers: Subtraction Method (Easy for Humans)

- Goal
  - Get the binary weights to add up to the decimal quantity
    - Work from left to right
    - (Right to left – may fill in 1s that shouldn't have been there – try it).

Desired decimal number: **12**

__	__	__	__	__	__	
32	16	8	4	2	1	
1	__	__	__	__	__	=32
32	16	8	4	2	1	too much
0	1	__	__	__	__	=16
32	16	8	4	2	1	too much
0	0	1	__	__	__	=8
32	16	8	4	2	1	ok, keep going
0	0	1	1	__	__	=8+4=12
32	16	8	4	2	1	DONE
0	**0**	**1**	**1**	**0**	**0**	answer
32	16	8	4	2	1	

Converting from Decimal to Binary Numbers: Subtraction Method (Easy for Humans)

- Subtraction method
  - To make the job easier (especially for big numbers), we can just subtract a selected binary weight from the (remaining) quantity
    - Then, we have a new remaining quantity, and we start again (from the present binary position)
    - Stop when remaining quantity is 0

Remaining quantity: 12

32	16	8	4	2	1	
1						32 is too much
32	16	8	4	2	1	
0	1					16 is too much
32	16	8	4	2	1	
0	0	1				**12 – 8 = 4**
32	16	8	4	2	1	
0	0	1	1			**4-4=0** DONE
32	16	8	4	2	1	
0	**0**	**1**	**1**	**0**	**0**	answer
32	16	8	4	2	1	

Converting from Decimal to Binary Numbers: Subtraction Method Example

- Q: Convert the number “23” from decimal to binary

A: Remaining quantity

Binary Number

23

0	0	0	0	0	0
32	16	8	4	2	1

23
-16
7

a

0	1	0	0	0	0
32	16	8	4	2	1

7
-4
3

0	1	0	1	0	0
32	16	8	4	2	1

8 is more than 7, can’t use

4
-2
1

0	1	0	1	1	0
32	16	8	4	2	1

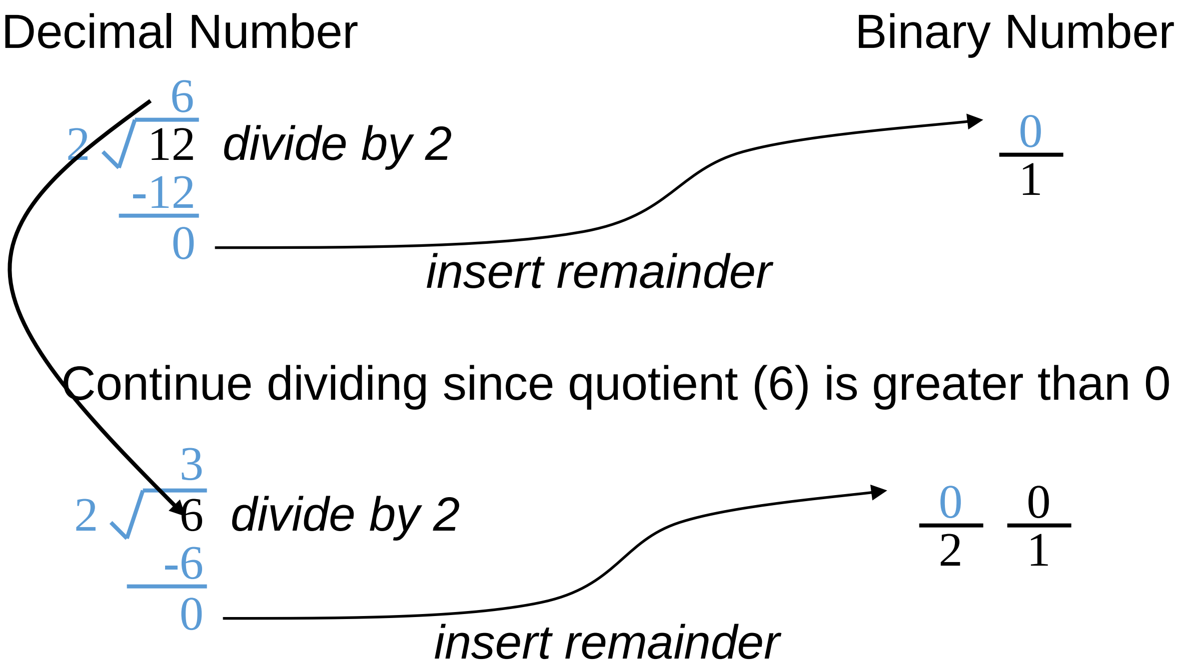
1
-1
0

0	1	0	1	1	1
32	16	8	4	2	1

Done! 23 in decimal is 10111 in binary.

Converting from Decimal to Binary Numbers: Division Method (Good for Computers)

- Divide decimal number by 2 and insert remainder into new binary number.
  - Continue dividing quotient by 2 until the quotient is 0.
- Example: Convert decimal number 12 to binary

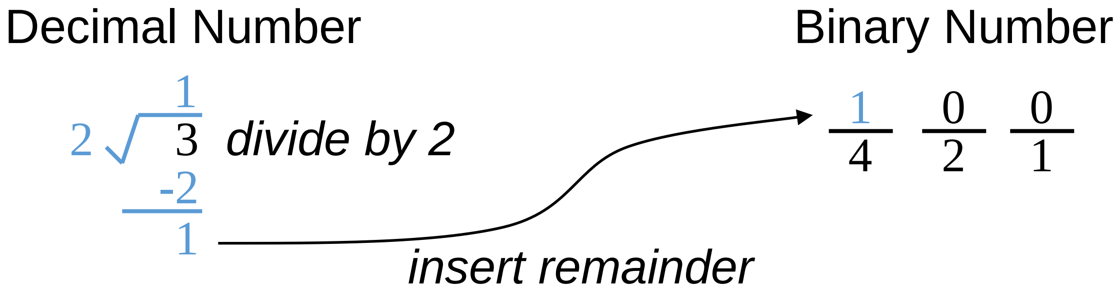


Continue dividing since quotient (6) is greater than 0

Continue dividing since quotient (3) is greater than 0

Converting from Decimal to Binary Numbers: Division Method (Good for Computers)

- Example: Convert decimal number 12 to binary (continued)



Decimal Number

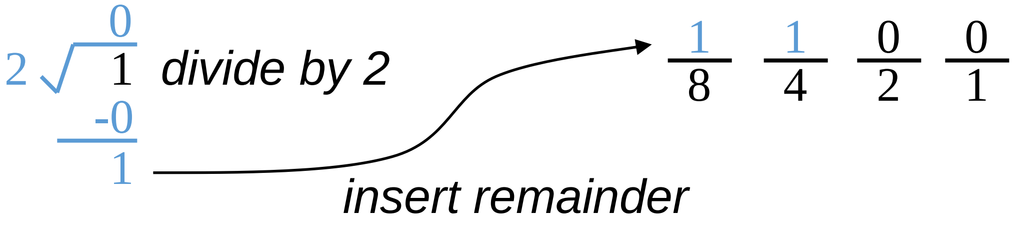
Binary Number

$2\overline{)\,3\,}$ = 1, -2, remainder 1 — *divide by 2*

insert remainder

$\frac{1}{4}\ \frac{0}{2}\ \frac{0}{1}$

Continue dividing since quotient (1) is greater than 0



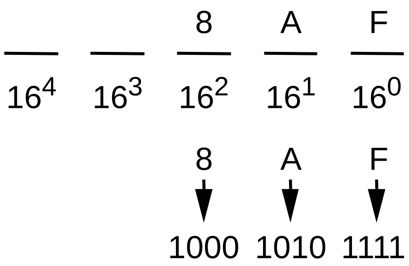
$2\overline{)\,1\,}$ = 0, -0, remainder 1 — *divide by 2*

insert remainder

$\frac{1}{8}\ \frac{1}{4}\ \frac{0}{2}\ \frac{0}{1}$

Since quotient is 0, we can conclude that 12 is 1100 in binary

Base Sixteen: Another Base Sometimes Used by Digital Designers

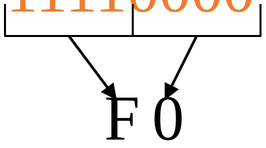


hex	binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

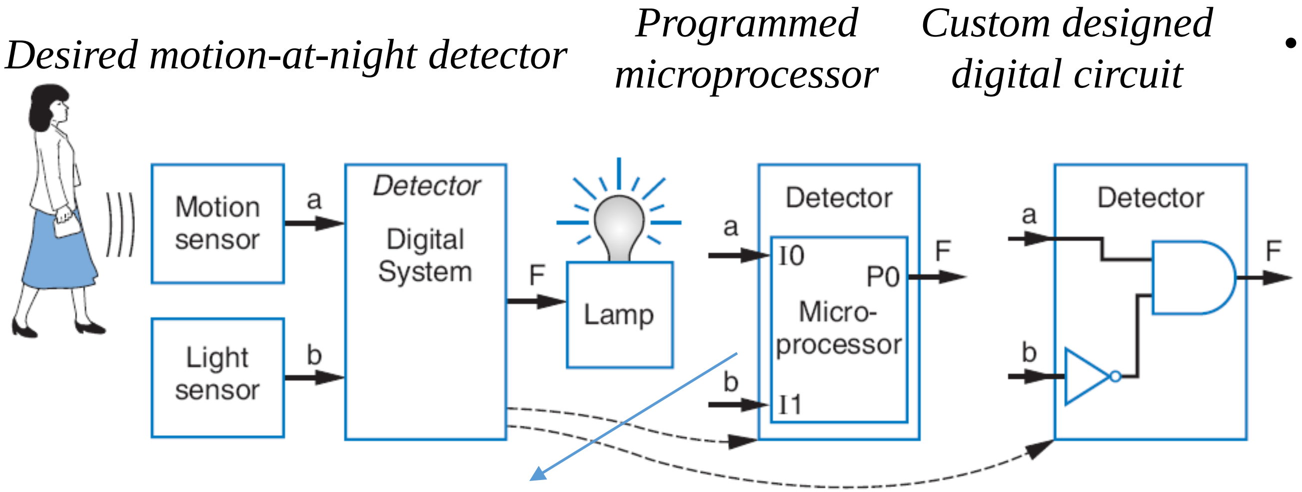
hex	binary
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

- Nice because each position represents four base two positions
  - Used as compact means to write binary numbers
- Known as ***hexadecimal***, or just ***hex***

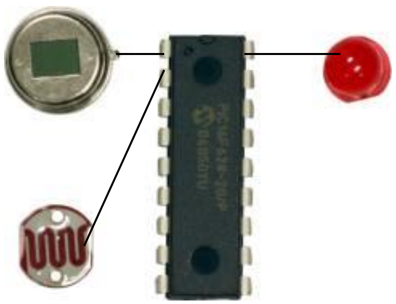
Q: Write 11110000 in hex

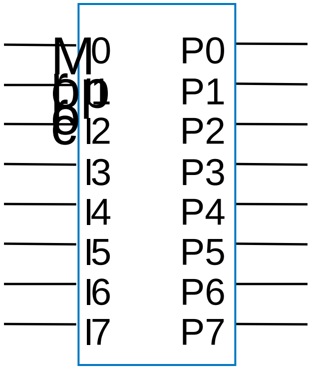


Implementing Digital Systems: Programming Microprocessors Vs. Designing Digital Circuits

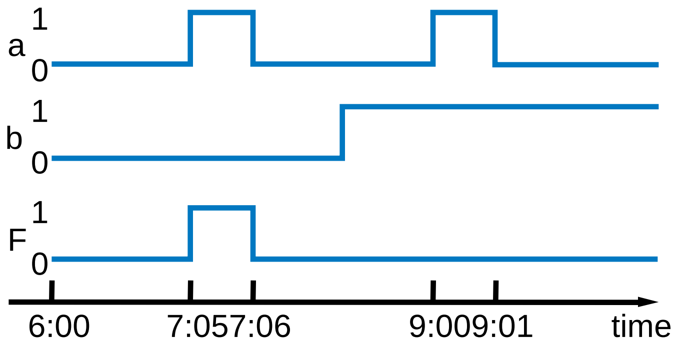


- Microprocessors a common choice to implement a digital system
  - Easy to program
  - Cheap (as low as $1)
  - Available now





```
void main()
{
   while (1) {
      P0 = I0 && !I1;
      // F = a and !b,
   }
}
```



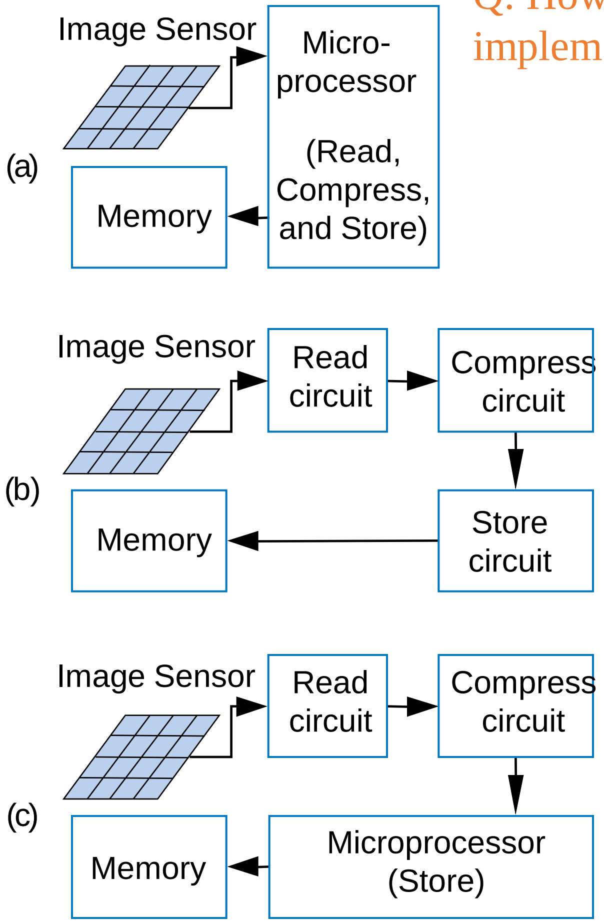
Digital Design: When Microprocessors Aren't Good Enough

- With microprocessors so easy, cheap, and available, why design a digital circuit?
  - Microprocessor may be too slow
  - Or too big, power hungry, or costly

Sample digital camera task execution times (in seconds) on a microprocessor versus a digital circuit:

Task	Microprocessor	Custom Digital Circuit
Read	5	0.1
Compress	8	0.5
Store	1	0.8

Q: How long for each implementation option?



5+8+1
=14 sec

.1+.5+.8
=1.4 sec

.1+.5+1
=1.6 sec

Good compromise

Chapter Summary

- Digital systems surround us
    - Inside computers
    - Inside huge variety of other electronic devices (embedded systems)
- Digital systems use 0s and 1s
    - Encoding analog signals to digital can provide many benefits
        - e.g., audio -- higher-quality storage/transmission, compression, etc.
    - Encoding integers as 0s and 1s: Binary numbers
- Microprocessors (themselves digital) can implement many digital systems easily and inexpensively
    - But often not good enough -- need custom digital circuits